Cheatsheet - Kubectl

Kubectl is a command line interface for running commands against Kubernetes clusters.

Installing

The kubectl version must be within one minor version difference of the Kubernetes cluster. For example, a v1.20 client should work with v1.19, v1.20, and v1.21 master.

Kubectl can be installed on Ubuntu, Debian, CentOS, RedHat operating systems.

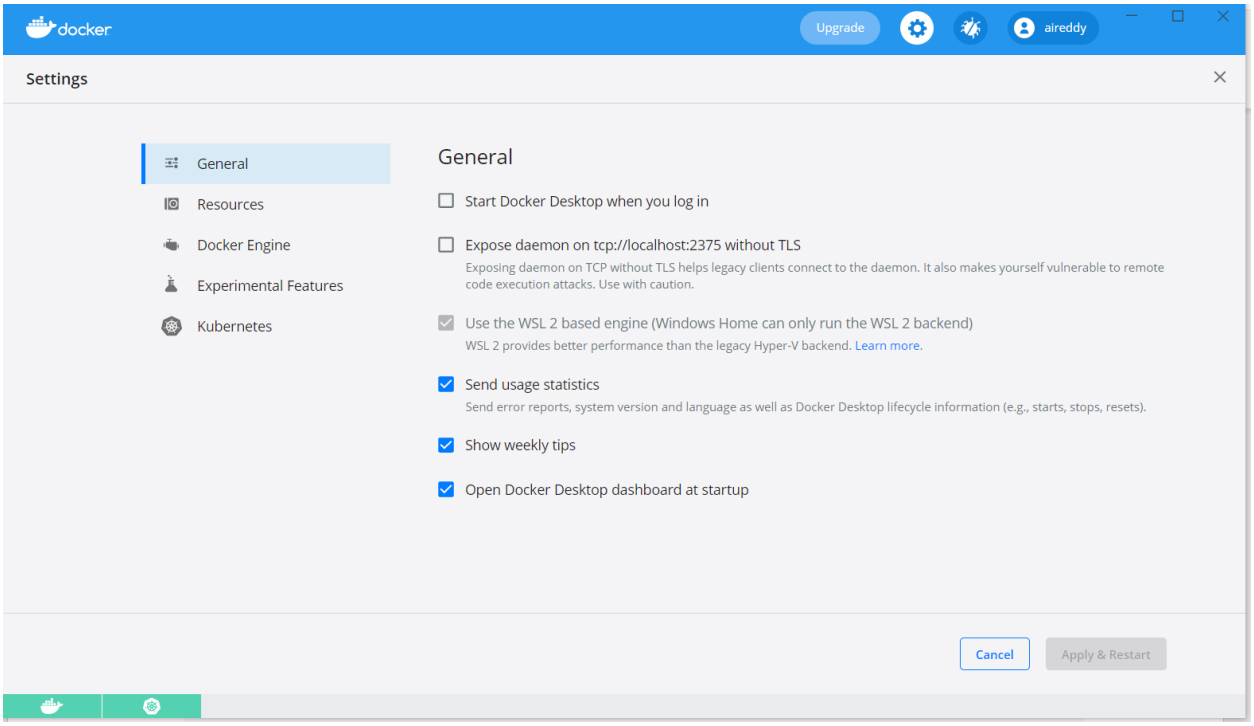
Install using native package management.

Assuming you already installed docker desktop for Windows and Enabled Kubernetes.

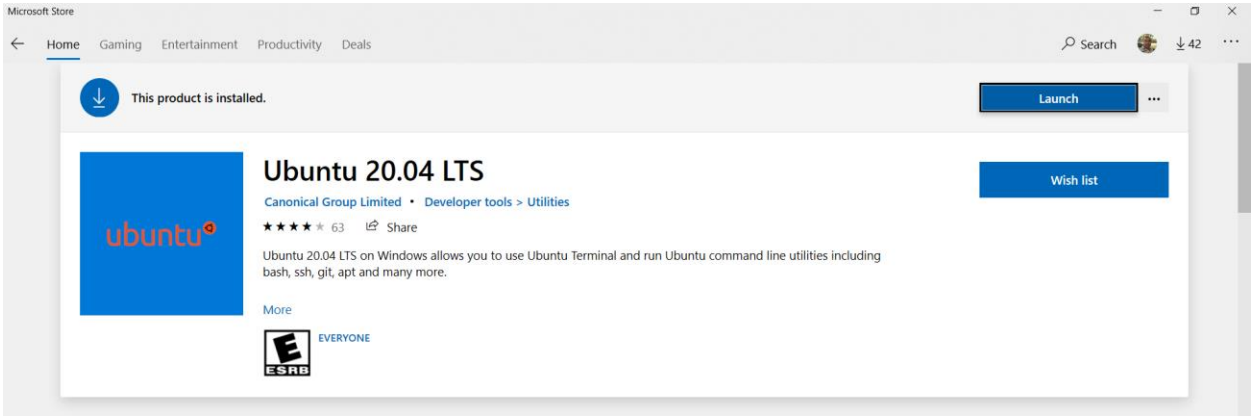
https://docs.docker.com/docker-for-windows/install/

https://hub.docker.com/editions/community/docker-ce-desktop-windows/

once you install docker desktop launch and enable Kubernetes from settings , you should see screen like below ..



Install Ubuntu app from Microsoft Appstore Ubuntu 20.4




```
mkdir .kube
vi config
copy C:\Users\****\.kube config content to config file in vi and save
```

Ubuntu / Debian

- Debian-based distributions
- Red Hat-based distributions

1. Update the `apt` package index and install packages needed to use the Kubernetes `apt` repository:

```
sudo apt-get update
sudo apt-get install -y apt-transport-https ca-certificates curl
```

4. Download the Google Cloud public signing key:

```
sudo curl -fsSLo /usr/share/keyrings/kubernetes-archive-keyring.gpg https://packages.cloud.google.com/apt/doc/apt-key.gpg
```

6. Add the Kubernetes `apt` repository:

```
echo "deb [signed-by=/usr/share/keyrings/kubernetes-archive-keyring.gpg] https://apt.kubernetes.io/ kubernetes-xenial main" | sudo tee /etc/apt/sources.list.d/kubernetes.list
```

8. Update `apt` package index with the new repository and install kubectl:

```
sudo apt-get update
sudo apt-get install -y kubectl
```

Now you can be able to run kubectl from ubuntu 20.4L

For further information about kubectl installation method, please refer to the Kubernetes documentation.

Completion

Kubectl autocomplete

BASH

```
source <(kubectl completion bash) # setup autocomplete in bash into the current shell, bash-completion package should be installed first.
echo "source <(kubectl completion bash)" >> ~/.bashrc # add autocomplete permanently to your bash shell.
```

You can also use a shorthand alias for `kubectl` that also works with completion:

```
alias k=kubectl
complete -F __start_kubectl k
```

Syntax

Kubectl is a powerful tool to manage each object on a Kubernetes cluster. The command has a simple and unique syntax to manage everything :

```
kubectl [command] [TYPE] [NAME] [flags]
```

- *command* : specifies the operation that you want to perform on one or more resources (create, get, describe, delete)
- *type* : specifies the resource type. Resource types are case-insensitive and you can specify the singular, plural, or abbreviated forms
- *name* : specifies the name of the resource. Names are case-sensitive. If the name is omitted, details for all resources are displayed
- *flags* : specifies optional flags.

Useful basic commands

Create

Create a resource from a file or from stdin.

```
Usage:
  kubectl create -f FILENAME [options]

Available Commands:
  clusterrole         Create a ClusterRole.
  clusterrolebinding  Create a ClusterRoleBinding for a particular ClusterRole
  configmap           Create a configmap from a local file, directory or literal value
  cronjob             Create a cronjob with the specified name.
  deployment          Create a deployment with the specified name.
  ingress             Create an ingress with the specified name.
  job                 Create a job with the specified name.
  namespace           Create a namespace with the specified name
  poddisruptionbudget Create a pod disruption budget with the specified name.
  priorityclass       Create a priorityclass with the specified name.
  quota               Create a quota with the specified name.
  role                Create a role with single rule.
  rolebinding         Create a RoleBinding for a particular Role or ClusterRole
  secret              Create a secret using specified subcommand
  service             Create a service using specified subcommand.
  serviceaccount      Create a service account with the specified name


# Create a pod using the data in pod.json.
kubectl create -f ./pod.json

# Create a pod based on the JSON passed into stdin.
cat pod.json | kubectl create -f -

# Edit the data in docker-registry.yaml in JSON using the v1 API format then create
the resource using the edited data.
kubectl create -f docker-registry.yaml --edit --output-version=v1 -o json

# Create all the resources avaibale in the folder
kubeclt create -f <folder_name>
```

Delete

Delete resources by filenames, stdin, resources and names, or by resources and label selector.

```
Usage:
  kubectl delete ([-f FILENAME] | [-k DIRECTORY] | TYPE [(NAME | -l label | --all)])
[options]

# Delete a pod using the type and name specified in pod.json.
kubectl delete -f ./pod.json

# Delete a pod based on the type and name in the JSON passed into stdin.
cat pod.json | kubectl delete -f -

# Delete pods and services with same names "baz" and "foo"
kubectl delete pod,service baz foo

# Delete pods and services with label name=myLabel.
kubectl delete pods,services -l name=myLabel

# Delete a pod with minimal delay
kubectl delete pod foo --now

# Force delete a pod on a dead node
kubectl delete pod foo --grace-period=0 --force

# Delete all pods
kubectl delete pods --all

#Delete all resources available in the folder.
kubectl delete -f <folder_name>
```

Edit

Edit a resource from the default editor.

```
Usage:
  kubectl edit (RESOURCE/NAME | -f FILENAME) [options]

# Edit the service named 'docker-registry':
kubectl edit svc/docker-registry

# Use an alternative editor
KUBE_EDITOR="nano" kubectl edit svc/docker-registry
```

```
# Edit the job 'myjob' in JSON using the v1 API format:
kubectl edit job.v1.batch/myjob -o json

# Edit the deployment 'mydeployment' in YAML and save the modified config in its
annotation:
kubectl edit deployment/mydeployment -o yaml --save-config
```

Expose

Expose a resource as a new Kubernetes service.

```
Usage:
  kubectl expose (-f FILENAME | TYPE NAME) [--port=port] [--protocol=TCP|UDP|SCTP] [-
-target-port=number-or-name]
[--name=name] [--external-ip=external-ip-of-service] [--type=type] [options]


# Create a service for a replicated nginx, which serves on port 80 and connects to
the containers on port 8000.
kubectl expose rc nginx --port=80 --target-port=8000

# Create a service for a replication controller identified by type and name specified
in "nginx-controller.yaml", which serves on port 80 and connects to the containers on
port 8000.
kubectl expose -f nginx-controller.yaml --port=80 --target-port=8000

# Create a service for a pod valid-pod, which serves on port 444 with the name
"frontend"
kubectl expose pod valid-pod --port=444 --name=frontend

# Create a second service based on the above service, exposing the container port
8443 as port 443 with the name "nginx-https"
kubectl expose service nginx --port=443 --target-port=8443 --name=nginx-https

# Create a service for a replicated streaming application on port 4100 balancing UDP
traffic and named 'video-stream'.
kubectl expose rc streamer --port=4100 --protocol=udp --name=video-stream

# Create a service for a replicated nginx using replica set, which serves on port 80
and connects to the containers on port 8000.
kubectl expose rs nginx --port=80 --target-port=8000

# Create a service for an nginx deployment, which serves on port 80 and connects to
the containers on port 8000.
kubectl expose deployment nginx --port=80 --target-port=8000

# Access Pod without exposing as service using kubectl --raw
kubectl get pod <pod_name> -o yaml|grep selfLink
kubectl get --raw <selfLink>:port/proxy/<filename>
```

Get

Display one or many resources.

```
Usage:
  kubectl get
[(-o|--output=)json|yaml|wide|custom-columns=...|custom-columns-file=...|go-
template=...|go-template-file=...|jsonpath=...|jsonpath-file=...]
(TYPE[.VERSION][.GROUP] [NAME | -l label] | TYPE[.VERSION][.GROUP]/NAME ...) [flags]
[options]


# List all pods.
kubectl get pods

# List all pods in ps output format with more information (such as node name).
kubectl get pods -o wide

# List a single replication controller with specified NAME in ps output format.
kubectl get replicationcontroller web

# List a single pod in JSON output format.
kubectl get -o json pod <pod-name>

# List a pod identified by type and name specified in "pod.yaml" in JSON output
format.
kubectl get -f pod.yaml -o json

# Return only the phase value of the specified pod.
kubectl get -o template pod/<pod-name> --template=

# List all replication controllers and services together in ps output format.
kubectl get rc,services

# List one or more resources by their type and names.
kubectl get rc/web service/frontend pods/<pod-name>

# List all resources with different types.
kubectl get all
```

Run

Create and run a particular image, possibly replicated.

```
# Usage:
kubectl run NAME --image=image [--env="key=value"] [--port=port] [--dry-
run=server|client] [--overrides=inline-json] [--command] -- [COMMAND] [args...]
[options]

# Start a single instance of nginx.
kubectl run nginx --image=nginx

# Start a single instance of hazelcast and let the container expose port 5701 .
```

```
kubectl run hazelcast --image=hazelcast --port=5701

# Start a single instance of hazelcast and set environment variables
"DNS_DOMAIN=cluster" and "POD_NAMESPACE=default" in the container.
kubectl run hazelcast --image=hazelcast --env="DNS_DOMAIN=cluster" --
env="POD_NAMESPACE=default"

# Start a single instance of hazelcast and set labels "app=hazelcast" and "env=prod"
in the container.
kubectl run hazelcast --image=nginx --labels="app=hazelcast,env=prod"

# Start a replicated instance of nginx.
kubectl run nginx --image=nginx --replicas=5

# Dry run. Print the corresponding API objects without creating them.
kubectl run nginx --image=nginx --dry-run

# Start a single instance of nginx, but overload the spec of the deployment with a
partial set of values parsed from JSON.
kubectl run nginx --image=nginx --overrides='{ "apiVersion": "v1", "spec": { ... } }'

# Start a pod of busybox and keep it in the foreground, don't restart it if it exits.
kubectl run -i -t busybox --image=busybox --restart=Never

# Start the nginx container using the default command, but use custom arguments (arg1
.. argN) for that command.
kubectl run nginx --image=nginx -- <arg1> <arg2> ... <argN>

# Start the nginx container using a different command and custom arguments.
kubectl run nginx --image=nginx --command -- <cmd> <arg1> ... <argN>

# Start the perl container to compute π to 2000 places and print it out.
kubectl run pi --image=perl --restart=OnFailure -- perl -Mbignum=bpi -wle 'print
bpi(2000)'

# Start the cron job to compute π to 2000 places and print it out every 5 minutes.
kubectl run pi --schedule="0/5 * * * ?" --image=perl --restart=OnFailure -- perl -
Mbignum=bpi -wle 'print bpi(2000)'
```

Set

Configure application resources.

```
Available Commands:
  env            Update environment variables on a pod template
  image          Update image of a pod template
  resources      Update resource requests/limits on objects with pod templates
  selector       Set the selector on a resource
  serviceaccount Update ServiceAccount of a resource
  subject        Update User, Group or ServiceAccount in a RoleBinding/ClusterRoleBinding
```

```
Usage:

  kubectl set SUBCOMMAND [options]

# Update deployment 'registry' with a new environment variable
kubectl set env deployment/registry STORAGE_DIR=/local

# List the environment variables defined on a deployments 'sample-build'
kubectl set env deployment/sample-build --list

# List the environment variables defined on all pods
kubectl set env pods --all --list

# Output modified deployment in YAML, and does not alter the object on the server
kubectl set env deployment/sample-build STORAGE_DIR=/data -o yaml

# Update all containers in all replication controllers in the project to have
ENV=prod
kubectl set env rc --all ENV=prod

# Import environment from a secret
kubectl set env --from=secret/mysecret deployment/myapp

# Import environment from a config map with a prefix
kubectl set env --from=configmap/myconfigmap --prefix=MYSQL_ deployment/myapp

# Remove the environment variable ENV from container 'c1' in all deployment configs
kubectl set env deployments --all --containers="c1" ENV-

# Remove the environment variable ENV from a deployment definition on disk and
# update the deployment config on the server
kubectl set env -f deploy.json ENV-

# Set some of the local shell environment into a deployment config on the server
env | grep RAILS_ | kubectl set env -e - deployment/registry

# Set a deployment's nginx container image to 'nginx:1.9.1', and its busybox
container image to 'busybox'.
kubectl set image deployment/nginx busybox=busybox nginx=nginx:1.9.1

# Update all deployments' and rc's nginx container's image to 'nginx:1.9.1'
kubectl set image deployments,rc nginx=nginx:1.9.1 --all

# Update image of all containers of daemonset abc to 'nginx:1.9.1'
kubectl set image daemonset abc *=nginx:1.9.1

# Print result (in yaml format) of updating nginx container image from local file,
without hitting the server
kubectl set image -f path/to/file.yaml nginx=nginx:1.9.1 --local -o yaml

# Set a deployments nginx container cpu limits to "200m" and memory to "512Mi"
kubectl set resources deployment nginx -c=nginx --limits=cpu=200m,memory=512Mi

# Set the resource request and limits for all containers in nginx
```

```
kubectl set resources deployment nginx --limits=cpu=200m,memory=512Mi --
requests=cpu=100m,memory=256Mi

# Remove the resource requests for resources on containers in nginx
kubectl set resources deployment nginx --limits=cpu=0,memory=0 --
requests=cpu=0,memory=0

# Print the result (in yaml format) of updating nginx container limits from a local,
without hitting the server
kubectl set resources -f path/to/file.yaml --limits=cpu=200m,memory=512Mi --local -o
yaml

# Set Deployment nginx-deployment's ServiceAccount to serviceaccount1
kubectl set serviceaccount deployment nginx-deployment serviceaccount1

# Print the result (in yaml format) of updated nginx deployment with serviceaccount
from local file, without hitting apiserver
kubectl set sa -f nginx-deployment.yaml serviceaccount1 --local --dry-run -o yaml
```

Useful deploy commands.

Autoscale

Creates an autoscaler that automatically chooses and sets the number of pods that run in a kubernetes cluste

```
Usage:
  kubectl autoscale (-f FILENAME | TYPE NAME | TYPE/NAME) [--min=MINPODS] --
max=MAXPODS [--cpu-percent=CPU] [options]


# Auto scale a deployment "foo", with the number of pods between 2 and 10, no target
CPU utilization specified so a default autoscaling policy will be used:
kubectl autoscale deployment foo --min=2 --max=10

# Auto scale a replication controller "foo", with the number of pods between 1 and 5,
target CPU utilization at 80%:
kubectl autoscale rc foo --max=5 --cpu-percent=80
```

Rollout

Manage the rollout of a resource.

```

Available Commands:
  history     View rollout history
  pause       Mark the provided resource as paused
  restart     Restart a resource
  resume      Resume a paused resource
  status      Show the status of the rollout
  undo        Undo a previous rollout
```

```
Usage:
  kubectl rollout SUBCOMMAND [options]

# Rollback to the previous deployment
kubectl rollout undo deployment/abc

# Check the rollout status of a daemonset
kubectl rollout status daemonset/foo

# View the rollout history of a deployment
kubectl rollout history deployment/abc

# View the details of daemonset revision 3
kubectl rollout history daemonset/abc --revision=3

# Mark the nginx deployment as paused. Any current state of
# the deployment will continue its function, new updates to the deployment will not
# have an effect as long as the deployment is paused.
kubectl rollout pause deployment/nginx

# Resume an already paused deployment
kubectl rollout resume deployment/nginx

# Watch the rollout status of a deployment
kubectl rollout status deployment/nginx

# Rollback to the previous deployment
kubectl rollout undo deployment/abc

# Rollback to daemonset revision 3
kubectl rollout undo daemonset/abc --to-revision=3

# Rollback to the previous deployment with dry-run
kubectl rollout undo --dry-run=true deployment/abc
```

Scale

Set a new size for a Deployment, ReplicaSet, Replication Controller, or StatefulSet.

```
Usage:
  kubectl scale [--resource-version=version] [--current-replicas=count] --
replicas=COUNT (-f FILENAME | TYPE NAME)
[options]

# Scale a replicaset named 'foo' to 3.
kubectl scale --replicas=3 rs/foo
```

```
# Scale a resource identified by type and name specified in "foo.yaml" to 3.
kubectl scale --replicas=3 -f foo.yaml

# If the deployment named mysql's current size is 2, scale mysql to 3.
kubectl scale --current-replicas=2 --replicas=3 deployment/mysql

# Scale multiple replication controllers.
kubectl scale --replicas=5 rc/foo rc/bar rc/baz

# Scale statefulset named 'web' to 3.
kubectl scale --replicas=3 statefulset/web
```

Useful cluster management commands

Cluster-info

Display addresses of the master and services with label kubernetes.io/cluster-service=true To further debug and diagnose cluster problems, use ‘kubectl cluster-info dump’.

```
# Print the address of the master and cluster services
kubectl cluster-info
```

Cordon / Uncordon

Mark node as (un)schedulable.

```
Usage:
  kubectl cordon NODE [options]

# Mark node "foo" as unschedulable.
kubectl cordon foo

Usage:
  kubectl uncordon NODE [options]
# Mark node "foo" as schedulable.
$ kubectl uncordon foo
```

Drain

Drain node in preparation for maintenance.

```
Usage:
  kubectl drain NODE [options]

# Drain node "foo", even if there are pods not managed by a ReplicationController,
ReplicaSet, Job, DaemonSet or StatefulSet on it.
$ kubectl drain foo --force
```

```
# As above, but abort if there are pods not managed by a ReplicationController,
ReplicaSet, Job, DaemonSet or StatefulSet, and use a grace period of 15 minutes.
$ kubectl drain foo --grace-period=90

#Drain node by ignoring Deamonsets
kubectl drain <node_name> --ignore-daemonsets
```

Taint

Update the taints on one or more nodes.

```
Usage:
  kubectl taint NODE NAME KEY_1=VAL_1:TAINT_EFFECT_1 ... KEY_N=VAL_N:TAINT_EFFECT_N
[options]

# Update node 'foo' with a taint with key 'dedicated' and value 'special-user' and
effect 'NoSchedule'.
# If a taint with that key and effect already exists, its value is replaced as
specified.
kubectl taint nodes foo dedicated=special-user:NoSchedule

# Remove from node 'foo' the taint with key 'dedicated' and effect 'NoSchedule' if
one exists.
kubectl taint nodes foo dedicated:NoSchedule-

# Remove from node 'foo' all the taints with key 'dedicated'
kubectl taint nodes foo dedicated-

# Add a taint with key 'dedicated' on nodes having label mylabel=X
kubectl taint node -l myLabel=X  dedicated=foo:PreferNoSchedule
```

Top

Display Resource (CPU/Memory/Storage) usage.

```
Usage:
  kubectl top [flags] [options]
Display Resource (CPU/Memory/Storage) usage.

 The top command allows you to see the resource consumption for nodes or pods.

 This command requires Metrics Server to be correctly configured and working on the
server.

Available Commands:
  node        Display Resource (CPU/Memory/Storage) usage of nodes
  pod         Display Resource (CPU/Memory/Storage) usage of pods

# Show metrics for all nodes
kubectl top node
```

```
# Show metrics for a given node
kubectl top node NODE_NAME

# Show metrics for all pods in the default namespace
kubectl top pod

# Show metrics for all pods in the given namespace
kubectl top pod --namespace=NAMESPACE

# Show metrics for a given pod and its containers
kubectl top pod POD_NAME --containers

# Show metrics for the pods defined by label name=myLabel
kubectl top pod -l name=myLabel
```

Useful troubleshooting and debugging commands.

Describe

Show details of a specific resource or group of resources.

```
Usage:
  kubectl describe (-f FILENAME | TYPE [NAME_PREFIX | -l label] | TYPE/NAME)
[options]

# Describe a node
kubectl describe nodes kubernetes-node-emt8.c.myproject.internal

# Describe a pod
kubectl describe pods/<pod-name>

# Describe a pod identified by type and name in "pod.json"
kubectl describe -f pod.json

# Describe all pods
kubectl describe pods

# Describe pods by label name=myLabel
kubectl describe po -l name=myLabel

# Describe all pods managed by the 'frontend' replication controller (rc-created pods
# get the name of the rc as a prefix in the pod the name).
kubectl describe pods frontend
```

Exec

Execute a command in a container.

```
# Get output from running 'date' from pod 123456-7890, using the first container by
default
kubectl exec 123456-7890 date

# Get output from running 'date' in ruby-container from pod 123456-7890
kubectl exec 123456-7890 -c ruby-container date

# Switch to raw terminal mode, sends stdin to 'bash' in ruby-container from pod
123456-7890
# and sends stdout/stderr from 'bash' back to the client
kubectl exec 123456-7890 -c ruby-container -i -t -- bash -il

# List contents of /usr from the first container of pod 123456-7890 and sort by
modification time.
# If the command you want to execute in the pod has any flags in common (e.g. -i),
# you must use two dashes (--) to separate your command's flags/arguments.
# Also note, do not surround your command and its flags/arguments with quotes
# unless that is how you would execute it normally (i.e., do ls -t /usr, not "ls -t
/usr").
kubectl exec 123456-7890 -i -t -- ls -t /usr
```

Logs

Print the logs for a container in a pod or specified resource. If the pod has only one container, the container name is optional.

```
Usage:
  kubectl logs [-f] [-p] (POD | TYPE/NAME) [-c CONTAINER] [options]
# Return snapshot logs from pod nginx with only one container
kubectl logs nginx

# Return snapshot logs for the pods defined by label app=nginx
kubectl logs -lapp=nginx

# Return snapshot of previous terminated ruby container logs from pod web-1
kubectl logs -p -c ruby web-1

# Begin streaming the logs of the ruby container in pod web-1
kubectl logs -f -c ruby web-1

# Display only the most recent 20 lines of output in pod nginx
kubectl logs --tail=20 nginx

# Show all logs from pod nginx written in the last hour
kubectl logs --since=1h nginx

# Return snapshot logs from first container of a job named hello
kubectl logs job/hello

# Return snapshot logs from container nginx-1 of a deployment named nginx
kubectl logs deployment/nginx -c nginx-1
```

Proxy

Creates a proxy server or application-level gateway between localhost and the Kubernetes API Server. It also allows serving static content over specified HTTP path. All incoming data enters through one port and gets forwarded to the remote Kubernetes API Server port, except for the path matching the static content path.

```
Usage:
  kubectl proxy [--port=PORT] [--www=static-dir] [--www-prefix=prefix] [--api-
prefix=prefix] [options]

# To proxy all of the kubernetes api and nothing else, use:
$ kubectl proxy --api-prefix=/

# To proxy only part of the kubernetes api and also some static files:
$ kubectl proxy --www=/my/files --www-prefix=/static/ --api-prefix=/api/
# The above lets you 'curl localhost:8001/api/v1/pods'.

# To proxy the entire kubernetes api at a different root, use:
$ kubectl proxy --api-prefix=/custom/
# The above lets you 'curl localhost:8001/custom/api/v1/pods'

# Run a proxy to kubernetes apiserver on port 8011, serving static content from
./local/www/
kubectl proxy --port=8011 --www=./local/www/

# Run a proxy to kubernetes apiserver on an arbitrary local port.
# The chosen port for the server will be output to stdout.
kubectl proxy --port=0
```

Useful advanced commands

Apply

Apply a configuration to a resource by filename or stdin. The resource name must be specified. This resource will be created if it does not exist yet. To use ‘apply’, always create the resource initially with either ‘apply’ or ‘create –save-config’.

```
Usage:
  kubectl apply (-f FILENAME | -k DIRECTORY) [options]


Available Commands:
  edit-last-applied Edit latest last-applied-configuration annotations of a resource/object
  set-last-applied  Set the last-applied-configuration annotation on a live object to match the contents
of a file.
  view-last-applied View latest last-applied-configuration annotations of a resource/object
```

```
# Apply the configuration in pod.json to a pod.
kubectl apply -f ./pod.json

# Apply the JSON passed into stdin to a pod.
cat pod.json | kubectl apply -f -

# Note: --prune is still in Alpha
# Apply the configuration in manifest.yaml that matches label app=nginx and delete
all the other resources that are not in the file and match label app=nginx.
kubectl apply --prune -f manifest.yaml -l app=nginx

# Apply the configuration in manifest.yaml and delete all the other configmaps that
are not in the file.
kubectl apply --prune -f manifest.yaml --all --prune-whitelist=core/v1/ConfigMap
```

Useful settings commands

label

Update the labels on a resource.

```
Usage:
  kubectl label [--overwrite] (-f FILENAME | TYPE NAME) KEY_1=VAL_1 ... KEY_N=VAL_N
[--resource-version=version]
[options]


# Update pod 'foo' with the label 'unhealthy' and the value 'true'.
kubectl label pods foo unhealthy=true

# Update pod 'foo' with the label 'status' and the value 'unhealthy', overwriting any
existing value.
kubectl label --overwrite pods foo status=unhealthy

# Update all pods in the namespace
kubectl label pods --all status=unhealthy

# Update a pod identified by the type and name in "pod.json"
kubectl label -f pod.json status=unhealthy

# Update pod 'foo' only if the resource is unchanged from version 1.
kubectl label pods foo status=unhealthy --resource-version=1

# Update pod 'foo' by removing a label named 'bar' if it exists.
# Does not require the --overwrite flag.
kubectl label pods foo bar-
```

Useful other commands

Config

Modify kubeconfig files using subcommands like “kubectl config set current-context my-context”.

```
The loading order follows these rules:

  1.  If the --kubeconfig flag is set, then only that file is loaded. The flag may
only be set once and no merging takes
place.
  2.  If $KUBECONFIG environment variable is set, then it is used as a list of paths
(normal path delimiting rules for
your system). These paths are merged. When a value is modified, it is modified in the
file that defines the stanza. When
a value is created, it is created in the first file that exists. If no files in the
chain exist, then it creates the
last file in the list.
  3.  Otherwise, ${HOME}/.kube/config is used and no merging takes place.

Available Commands:
  current-context Displays the current-context
  delete-cluster  Delete the specified cluster from the kubeconfig
  delete-context  Delete the specified context from the kubeconfig
  delete-user     Delete the specified user from the kubeconfig
  get-clusters    Display clusters defined in the kubeconfig
  get-contexts    Describe one or many contexts
  get-users       Display users defined in the kubeconfig
  rename-context  Renames a context from the kubeconfig file.
  set             Sets an individual value in a kubeconfig file
  set-cluster     Sets a cluster entry in kubeconfig
  set-context     Sets a context entry in kubeconfig
  set-credentials Sets a user entry in kubeconfig
  unset           Unsets an individual value in a kubeconfig file
  use-context     Sets the current-context in a kubeconfig file
  view            Display merged kubeconfig settings or a specified kubeconfig file

Usage:
  kubectl config SUBCOMMAND [options]
```

```
# Display the current-context
kubectl config current-context

# Delete the minikube cluster
kubectl config delete-cluster minikube

# Delete the context for the minikube cluster
kubectl config delete-context minikube

# List the clusters kubectl knows about
kubectl config get-clusters

# List the context kubectl knows about
kubectl config get-contexts

# Rename the context 'old-name' to 'new-name' in your kubeconfig file
kubectl config rename-context old-name new-name
```

```
# Set only the server field on the e2e cluster entry without touching other values.
kubectl config set-cluster e2e --server=https://1.2.3.4

# Embed certificate authority data for the e2e cluster entry
kubectl config set-cluster e2e --certificate-authority=~/.kube/e2e/kubernetes.ca.crt

# Disable cert checking for the dev cluster entry
kubectl config set-cluster e2e --insecure-skip-tls-verify=true

# Set the user field on the gce context entry without touching other values
kubectl config set-context gce --user=cluster-admin

# Use the context for the minikube cluster
kubectl config use-context minikube
```

Version

Print the client and server version information for the current context.

```
Usage:
  kubectl version [flags] [options]

# Print the client and server versions for the current context
kubectl version
```

Kubernetes documentation

To go further in the management of Kubectl, please refer to these documentations :

- Official Kubernetes overview of Kubectl command line
- Official Kubernetes documentation to install Kubectl command line
- Official Kubectl commands details